

## (Часть 2.1.3) Практика лабиринт DB

Лабиринт.

Сначала немного теории...

Поиск кратчайшего или оптимального пути через лабиринт распространенная в информатике задача поиска. Самые популярные алгоритмы поиска:

- в ширину
- в глубину
- алгоритм A\* (реализовать по желанию)

Представим лабиринт в виде двумерной сети ячеек `Cell`. Ячейка может быть в состояниях:

EMPTY – ' ' (пустая ячейка)

BLOCKED – 'X' или '■' (препятствие)

START – 'S' (стартовая ячейка)

END – 'E' (целевая ячейка)

PATH – '\*' или '□' (занятая ячейка найденным путем)

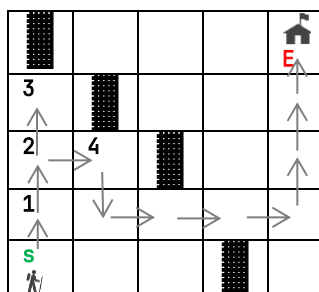
Лабиринт может иметь вид:

0	1	2	3	4	5	6	7	8	9	10
1	S			■			■			
2				■						
3							■			■
4			■							■
5	■	■								■
6				■		■	■			
7	■			■						
8		■								
9	■			■		■	■			
10		■								E

Как видите, чтобы пройти из точки S в точку E, необходимо обходить препятствия и в случае тупика возвращаться назад на шаг назад и делать попытку поиска снова.

В результате поиска решений как пройти, возможны варианты поиска:

1. Поиск в глубину (depth-first search, DFS) – поиск, который заходит насколько возможно глубоко и если процесс зайдет в тупик, то вернуться к последней точке принятия решения. Например, искомый путь может выглядеть так (ячейка 2 точка принятия решения):

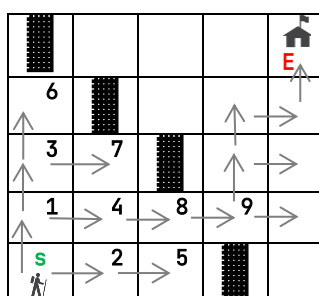


Поиск в глубину проходит по непрерывному пути вглубь, пока не дойдет до препятствия и не будет вынужден вернуться к последней точке принятия решения.

Алгоритм поиска в глубину опирается на структуру данных, известную как стек.

2. Путь прохода по лабиринту, найденного с помощью поиска в глубину, кажется неестественным. Обычно это не самый короткий путь.

Поиск в ширину (breadth-first search, BFS) всегда находит кратчайший путь, просматривая на каждой итерации поиска ближайшие ячейки по отношению к исходной. Однако, часто поиск в глубину позволяет найти решение быстрее, чем поиск в ширину.



При поиске в ширину сначала просматриваются ближайшие к начальной позиции элементы

Алгоритм поиска в ширину идентичен алгоритму поиска в глубину, только область поиска не стек, а очередь.

3. Поиск по алгоритму  $A^*$ , стремится найти кратчайший путь от начальной к конечной точке. При поиске  $A^*$  используется объединение функции затрат и эвристической функции, что позволяет сосредоточить поиск на путях, которые, скорее всего, быстро приведут к цели. Функция затрат  $g(n)$  проверяет затраты, необходимые для того, чтобы добраться

до определенного состояния. В случае с простым лабиринтом это может быть количество шагов, которые пришлось бы пройти, чтобы добраться до конечной ячейки. Эвристическая функция  $h(n)$  позволяет оценить затраты, необходимые для того, чтобы из заданной ячейки достичь целевую. Можно доказать, что если  $h(n)$  – допустимая эвристическая функция, то найденный конечный путь будет оптимальным. Допустимая эвристическая функция – это функция, которая никогда не переоценивает затраты на достижение цели. На двумерной плоскости пример такой эвристической функции – расстояние по прямой линии, поскольку прямая линия – всегда кратчайший путь. Общие затраты для любого рассматриваемого состояния определяются функцией  $f(n)$ , которая является простым объединением  $g(n)$  и  $h(n)$ . В сущности,  $f(n)=g(n)+h(n)$ . При выборе следующей рассматриваемой ячейки из области поиска алгоритм  $A^*$  выбирает ячейку с наименьшим  $f(n)$ . Именно этим он отличается от алгоритмов BFS и DFS.

Чтобы выбрать из области поиска состояние с наименьшим  $f(n)$ , при поиске  $A^*$  в качестве структуры данных используется очередь с приоритетом для данной области поиска. В очереди с приоритетом элементы сохраняются во внутренней последовательности, так что первый извлеченный элемент – это элемент с наивысшим приоритетом. (В нашем случае наиболее приоритетный элемент с наименьшим значением  $f(n)$ .) Обычно это означает задействование внутри очереди бинарной кучи, что дает сложность  $O(\lg n)$  для помещения в очередь и извлечения из нее.

Эвристика – это интуитивное представление о том, как решить задачу. В случае прохода по лабиринту эвристика стремится выбрать в нем лучшую точку для поиска следующей точки в желании добраться до цели. Эвристика – это обоснованное предположение о том, какие узлы из области поиска находятся ближе всего к цели. Если эвристика, используемая при поиске по алгоритму  $A^*$ , дает точный относительный результат и допустима (никогда не переоценивает расстояние), то  $A^*$  составит кратчайший путь. Эвристика, которая вычисляет меньшие значения, в

итоге дает поиск по большему количеству ячеек, тогда как эвристика, значение которой ближе к точному реальному расстоянию (но не больше него, иначе эвристика будет недопустимой), выполняет поиск по меньшему количеству ячеек. Следовательно, идеальная эвристика максимально приближается к реальному расстоянию, но не превышает его.

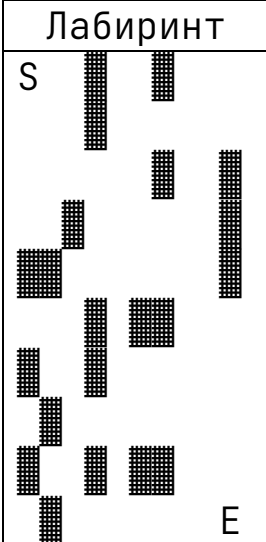
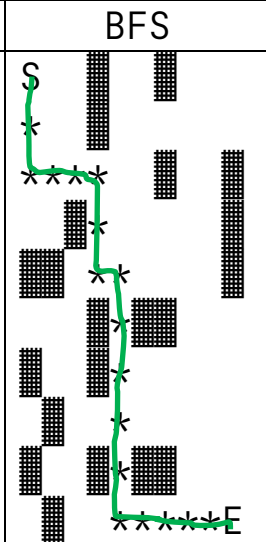
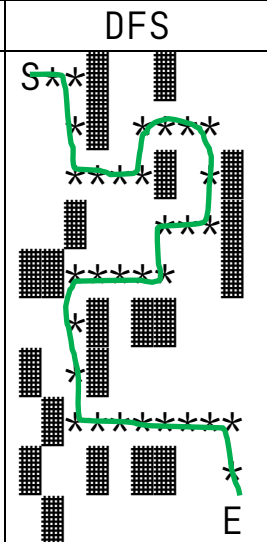
Теперь задание.

Задание из двух частей.

1) Сгенерировать сетку лабиринта и случайным образом заполнить блокирующими ячейками. Однако, сгенерированный лабиринт должен быть достаточно разреженным, чтобы почти всегда существовал путь от заданного начального до конечного местоположения. Функции генерирующей лабиринт, точно задавать степень разреженности. По умолчанию значение составляет 20 % заблокированных ячеек.

Вывести на экран лабиринт.

2) Реализовать, как описано выше алгоритмы DFS и BFS. Прохождение алгоритма отобразить на экране. Например:

Лабиринт	BFS	DFS
		

Зеленая дорожка просто для показа пути.

Лабиринт и пройденный путь записать в БД, с возможностью его восстановления. Обязательно! Запись в БД вести в процессе прохождения.

**Проходить по диагонали нельзя.**